3 Lab (1): SimpleNN for CIFAR-10 classification (15+4 pts)

Just like in HW1, here we start with a simple CNN architecture which we term as SimpleNN. It is composed of 2 CONV layers, 2 POOL layers, and 3 FC layers. The detailed structure of this model is shown in Table 1.

Name	Type	Kernel size	depth/units	Activation	Strides
Conv 1	Convolution	5	8	ReLU	1
MaxPool	MaxPool	2	N/A	N/A	2
Conv 2	Convolution	3	16	ReLU	1
MaxPool	MaxPool	2	N/A	N/A	2
FC1	Fully-connected	N/A	120	ReLU	N/A
FC2	Fully-connected	N/A	84	ReLU	N/A
FC3	Fully-connected	N/A	10	None	N/A

Table 1: SimpleNN structure. No padding is applied on both convolution layers. A flatten layer is required before FC1 to reshape the feature.

In this lab, beyond model implementation, you will learn to set up the whole training pipeline and actually train a classifier to perform image classification on the CIFAR-10 dataset [1]. CIFAR-10 is one of the most famous/popular benchmarks for image recognition/classification. It consists of 10 categories (e.g., bird, dog, car, airplane) with 32x32 RGB images. You may go to the official website for more information `https://www.cs.toronto.edu/~kriz/cifar.html`.

In this assignment, please refer to Jupyter Notebook `simplenn-cifar10.ipynb` for detailed instructions on how to construct a training pipeline for SimpleNN model. **Note, remember to unzip the provided `tools.zip` to your workspace before getting started.**

(a) (2 pts) As a sanity check, we should verify the implementation of the SimpleNN model at **Step 0**. Determine how you can check whether the model is implemented correctly, then check it.

Hints: 1) Consider creating dummy inputs that are of the same size as CIFAR-10 images, passing them through the model, and see if the model's outputs are of the correct shape. 2) Count the total number of parameters of all CONV/FC layers and see if it meets your expectation.

(b) (2 pts) Data preprocessing is crucial to enable successful training and inference of DNN models. Specify the preprocessing functions at **Step 1** and briefly discuss what operations you use and why.

(c) (2 pts) During the training, we need to feed data to the model, which requires an efficient data loading process. This is typically achieved by setting up a `dataset` and a `dataloader`. Please go to **Step 2** and build the actual training/validation datasets and dataloaders. Note, instead of using the `CIFAR10` dataset class from `torchvision.datasets`, here you are asked to use our own CIFAR-10 dataset class, which is imported from `tools.dataset`. As for the dataloader, we encourage you to use `torch.utils.data.DataLoader`.

(d) (2 pts) Go to **Step 3** to deploy the SimpleNN model on GPUs for efficient training. How can you verify that your model is indeed deployed on GPU? *Hint: use* `nvidia-smi` *command in the terminal*

(e) (2 pts) Loss functions are used to encode the learning objective. Now, we need to define this problem's loss function as well as the optimizer which will update our model's parameters to minimize the loss. In **Step 4**, please fill out the loss function and optimizer part.

(f) (2 pts) Follow the instructions in **Step 5** to set up the training process of SimpleNN on the CIFAR-10 dataset.

(g) (3 pts) Start training with the provided hyperparameter setting. What is the initial loss value before you conduct *any* training step? How is it related to the number of classes in CIFAR-10? What can you observe from **training accuracy** and **validation accuracy**? Do you notice any problems with the current training pipeline?

(h) (**Bonus**, 4 pts) Currently, we do not decay the learning rate during the training. Try to decay the learning rate (you may play with the `DECAY_EPOCHS` and `DECAY` hyperparameters in Step 5). What can you observe compared with no learning rate decay?

At the end of Lab 1, we expect at least 65% validation accuracy if all the steps are completed properly. You are required to submit the completed version of `simplenn-cifar10.ipynb` for Lab (1).

4 Lab (2): Improving the training pipeline (35+6 pts)

In Lab (1), we develop a simplified training pipeline. To obtain better training result, we will improve the training pipeline by employing data augmentation, improving the model design, and tuning the hyperparameters.

Before start, please duplicate the notebook in Lab (1) and name it as `simplenn-cifar10-dev.ipynb`, and work on the new notebook. You goal is to reach at least 70% validation accuracy on the CIFAR-10 dataset.

(a) (6 pts) Data augmentation techniques help combat overfitting. A typical strategy for CIFAR classification is to combine 1) *random cropping* with a *padding* of 4 and 2) *random flipping*. Train a model with such augmentation. How is the validation accuracy compared with the one without augmentation? **Note that in the following questions we all use augmentation. Also remember to reinitialize the model whenever you start a new training!**

(b) (15 pts) Model design is another important factor in determining performance on a given task. Now, modify the design of SimpleNN as instructed below:

   i. (5 pts) Add a batch normalization (BN) layer after each convolution layer. Compared with no BN layers, how does the best validation accuracy change?

   ii. (5 pts) Use empirical results to show that batch normalization allows a larger learning rate.

   iii. (5 pts) Implement Swish [2] activation on you own, and replace all of the ReLU activations in SimpleNN to Swish. Train the model with BN layers and a learning rate of 0.1. Does Swish outperform ReLU?

(c) (14 pts) Hyperparameter settings are very important and can have a large impact on the final model performance. Based on the improvements that you have made to the training pipeline thus far (with data augmentation and BN layers), tune some of the hyperparameters as instructed below:

   i. (7 pts) Apply different learning rate values: 1.0, 0.1, 0.05, 0.01, 0.005, 0.001, to see how the learning rate affects the model performance, and report results for each. Is a large learning rate beneficial for model training? If not, what can you conclude from the choice of learning rate?

   ii. (7 pts) Use different L2 regularization strengths of 1e-2, 1e-3, 1e-4, 1e-5, and 0.0 to see how the L2 regularization strength affects the model performance. In this problem use a learning rate of 0.01. Report the results for each regularization strength value. What did you expect? Are the results what you expected?

   iii. (Bonus, 6 pts) Switch the regularization penalty from L2 penalty to L1 penalty and train with the default hyperparameters. *Hint: This means you may not use the* `weight_decay` *parameter in PyTorch builtin optimizers, as it does not support L1 regularization. Instead, you need to add L1 penalty as a part of the loss function.* Compare the distribution of weight parameters after L1/L2 regularization. Describe your observations, are they what you expected? Why or why not?

Up to now, you shall have an improved training pipeline for CIFAR-10. Remember, you are required to submit `simplenn-cifar10-dev.ipynb` for Lab (2).

5 Lab (3): Advanced CNN architectures (20 pts)

The improved training pipeline for SimpleNN developed in Lab (2) still has limited performance. This is mainly because the SimpleNN has a rather small capacity (learning capability) for the CIFAR-10 task. Thus, in this lab, we replace the SimpleNN model with a more advanced ResNet [3] architecture. We expect to see much higher accuracy on CIFAR-10 when using ResNets. **Here, you may duplicate your jupyter notebook for Lab (2) as `resnet-cifar10.ipynb` to serve as a starting point.**

(a) (8 pts) Implement the ResNet-20 architecture by following Section 4.2 of the ResNet paper [3]. This lab is designed to have you learn how to implement a DNN model yourself, **so do NOT borrow any code from online resource.**

(b) (12 pts) Tune your ResNet-20 model to reach an accuracy of higher than 90% on the validation dataset. You may use all of the previous techniques that you have learned so far, including data augmentations, hyperparameter tuning, learning rate decay, etc. Training the model longer is also essential to obtaining good performance. You should be able to achieve >90% validation accuracy with a maximum of 200 epochs. **Remember to save your trained model during the training!!!** Check out this tutorial `https://pytorch.org/tutorials/beginner/saving_loading_models.html` on model saving/loading.

Note: We will grade this task by evaluating your trained model on the holdout testing dataset (which you do not have any labels). **After your ResNet-20 model is trained, you need to make predictions on test data, and save the predictions into the `predictions.csv` file**. Please utilize the given notebook, `save_test_predictions.ipynb`, to save your predictions in required format. The saved file should look like the provided example `sample_predictions.csv`. Upon submission, we will directly compare your predicted labels with the ground-truth labels to compute your score.

After completing Lab (3), you are required to submit `resnet-cifar10.ipynb` and your prediction results `predictions.csv`.

Info: Additional requirements:

- **DO NOT** train on the test set or use pretrained models to get unfair advantage. We have conducted a special preprocessing on the original CIFAR-10 dataset. As we have tested, "cheating" on the full dataset will give only 6% accuracy on our final test set, which means being unsuccessful in this assignment.
- **DO NOT** copy code directly online or from other classmates. We will check it! The result can be severe if your codes fail to pass our check.

Info: As this assignment requires significant computing resources (GPUs), we suggest:

- Plan your work in advance and start early. We will **NOT** extend the deadline because of the unavailability of computing resources.
- Be considerate and kill Jupyter Notebook instances when you do not need them.
- **DO NOT** run your program forever. Please follow the recommended/maximum training budget in each lab.

References

[1] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[2] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.